systems project

individual project
NoSQL, in c/c++

research project

groups of three
NoSQL, Neural Networks
Periodic Table of Data Structures

only open to cs165 students
(but one can transition after systems P)

semester project: due in the end of semester + a midway check in (early March, 10%)
projects in collaboration and supported by:

Released in ~Week 3-4
Quick summary: 2 reviews per week as of week 3 (or 4)
1 presentation in semester
individual systems or group research project (or both)
Quick summary: 2 reviews per week as of week 3 (or 4)
1 presentation in semester
individual systems or group research project (or both)

Check out: syllabus, preparation readings, project 0, systems project, online sections

http://daslab.seas.harvard.edu/classes/cs265/
learning outcome

fundamental of storage
data structures, SQL, NoSQL, Big Data, Neural Networks, Statistics, Data Science

paper readings from all these areas (as of week 3-4)
learning outcome
fundamental of storage

- data structures, SQL, NoSQL, Big Data, Neural Networks, Statistics, Data Science

  paper readings from all these areas (as of week 3-4)

software engineering  data-driven startup  research
Get familiar with the very basics of traditional database architectures:

Get familiar with very basics of modern database architectures:

Get familiar with the very basics of modern large scale systems:

The Periodic Table of Data Structures.
Stratos Idreos, Kostas Zoumpatianos, Manos Athanassoulis, Niv Dayan, Brian Hentschel, Michael S. Kester, Demi Guo, Lukas Maas, Wilson Qin, Abdul Wasay, Yiyou Sun. IEEE Data Engineering Bull. Sep, 2018
questions on logistics?
interaction: in and out of class

There is no such thing as a wrong question/answer!!!!
Research/Systems Intro

~3-4 lectures: Periodic Table, Calculator, NoSQL. Then Statistics, Neural Networks
GET $N$ EXPERT DESIGNERS
GIVE THEM $T$ TIME
HOPE FOR THE BEST
GET $N$ EXPERT DESIGNERS
GIVE THEM $T$ TIME
HOPE FOR THE BEST
THE HIPPO METHOD
"HIGHEST PAID PERSON'S OPINION"
standard “solution”

expose knobs
1 design/research skills do not scale
2 no one knows everything out there

NoSQL storage

The log-structured merge-tree (LSM-tree)
2 no one knows everything out there

NoSQL storage

P. O’Neil, E. Cheng, D. Gawlick, E, O’Neil
The log-structured merge-tree (LSM-tree)
workload
layout
design
h/w
without coding or accessing the h/w
without coding or accessing the h/w

workload

layout design

h/w

algorithms

performance

what-if design
What if I add bloom filters to my B-tree?

accessing the h/w

what-if design
What if I add bloom filters to my B-tree?

What if I add feature X that brings 60% more writes?
What if I need to reduce memory by 50%?

What if I add bloom filters to my B-tree?

What if add feature X that brings 60% more writes?
<table>
<thead>
<tr>
<th>What if I need to reduce memory by 50%?</th>
<th>What if add feature X that brings 60% more writes?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Should I buy new hardware X?</td>
<td>What if I add bloom filters to my B-tree?</td>
</tr>
<tr>
<td>Which workload breaks my system?</td>
<td></td>
</tr>
<tr>
<td>Cost in Amazon Cloud?</td>
<td></td>
</tr>
</tbody>
</table>
Research Projects:
What if design for Data Structure and Neural Networks in general and Key-value stores in particular (also for Systems Project)
Rob Tarjan, Turing Award 1986

“IS THERE A CALCULUS OF DATA STRUCTURES by which one can choose the appropriate representation and techniques for a given problem?” (SIAM, 1978)

[P vs NP, average case, constant factors vs asymptotic, low bounds]
How many and which structures are possible?
How many and which structures are possible?

Can we compute performance w/o coding?
EVERY DESIGN: 1 A SET OF CONCEPTS

2 EXISTING OR NEW CONCEPTS

3 ALL GOOD IDEAS IN THE 60s?

INDEX

scan, random access
binary search

metadata, model,
function, filters

physical layout,
e.g., partitioning

DATA

A SET OF CONCEPTS

EVERY DESIGN:

EXISTING OR
NEW CONCEPTS

ALL GOOD IDEAS IN THE 60s?

DATA

INDEX

scan, random access
binary search

metadata, model,
function, filters

physical layout,
e.g., partitioning

A SET OF CONCEPTS
EVERY DESIGN: 1 A SET OF CONCEPTS

2 EXISTING OR NEW CONCEPTS

3 ALL GOOD IDEAS IN THE 60s?

EXISTING

NEW
(ALMOST) ALL DESIGNS ARE A COMBINATION/TUNING OF EXISTING CONCEPTS

1. A SET OF CONCEPTS
2. EXISTING OR NEW CONCEPTS
3. ALL GOOD IDEAS IN THE 60s?
the ultimate action

I fear nothing

Nikos Kazantzakis, philosopher

researcher in action
“there is no pizza recipe
he who thinks there is a pizza recipe is no pizza maker”

Franco Pepe
what is creativity?

Plato
Leibniz
Nikos Kazantzakis, philosopher

I fear nothing. I am the most free form of action is for ultimate theory after all. Hope is for nothing holy holy free.
action is the most holy form of ultimate theory

I hope for nothing
I fear nothing
I am free

Nikos Kazantzakis, philosopher
Nikos Kazantzakis, philosopher

action is the ultimate theory

most holy form of

NEW

I hope for nothing
I fear nothing
I am free
action is the ultimate holy form of theory

NEW & BRILLIANT

I hope for nothing
I fear nothing
I am free

Nikos Kazantzakis, philosopher
CEREAL MILK PANNA COTTA
non obvious valid combinations

milk + cream + sugar + vanilla/lemon
**fundamental** building blocks

**properties** when combined
fundamental building blocks

properties when combined
fundamental building blocks properties when combined
trial and error
FIRST PRINCIPLE: design concept that does not break further
FIRST PRINCIPLE: design concept that does not break further

MAP LAYOUT FIRST
FIRST PRINCIPLE: design concept that does not break further

MAP LAYOUT FIRST

KNOWN DESIGNS

OPEN QUESTIONS
NoSQL Key-value Stores

machine learning  social media
smart homes  web browsers
phones  web-based apps
security  health devices
graphs  analytics
insert (key-value)
insert (key-value)
DISK MEMORY

Level 1
Level 1

MEMORY

DISK
insert (key-value)
MEMORY

DISK

Level 2
insert (key-value)
insert (key-value)

buffer

Level 1

Level 2

Level 3

...

Level N
insert (key-value)

buffer

Level 1

Level 2

Level 3

... 

Level N

MEMORY

DISK

pages

SSTables
tiered

leveled

sorted
[1,0,0,1,1,1] hash fun.
bloom filters

[min-max] /page
fence pointers

buffer

Level 1

Level 2

Level 3

Level N

MEMORY

DISK

pages

SSTables

leveled

tiered

sorted

hash fun.

fence pointers

[1,0,0,1,1,1]

[min-max]/page

bloom filters
get (key)

buffer

[1,0,0,1,1,1] hash fun.
bloom filters

[min-max]/page

disks

MEMORY

DISK

pages

SSTables

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max]/page

bloom filters

fence pointers

get (key)

Level 1

Level 2

Level 3

... 

Level N
get (key)

buffer

Level 1

Level 2

Level 3

... 

Level N

MEMORY

DISK

SSTables

pages

[1,0,0,1,1,1] hash fun.

[bloom filters]

[min-max] /page

[fence pointers]

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.

[1,0,0,1,1,1,1] hash fun.
DISK
MEMORY

pages

SSTables

[1,0,0,1,1,1]
hash fun.

bloom
filters

[min-max]
/fpage

fence
pointers

[0]
/min-max

.../page

get (key)

buffer

Level 1

Level 2

Level 3

.../page

Level N

leveled
tiered

sorted

hash fun.
get (key)

buffer

Level 1

Level 2

Level 3

... Level N

bloom filters

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

MEMORY

DISK

pages

SSTables

tiered

leveled

sorted

hash fun./page
get (key)

buffer

Level 1

Level 2

Level 3

Level N

DISK

MEMORY

SSTables

pages

tiered

leveled

sorted

bloom filters

fence pointers

[min-max] /page

[1,0,0,1,1,1] hash fun.
bloom filters
fence pointers

buffer

Level 1

Level 2

Level 3

...  

Level N

MEMORY
DISK

pages
SSTables

[1,0,0,1,1,1] hash fun.
[min-max] /page

hash fun.
fence pointers

[0,0,0,1,1,1]

[min-max] /page
- bloom filters
- fence pointers
- [1,0,0,1,1,1] hash fun.
- [min-max] /page

- buffer
- Level 1
- Level 2
- Level 3
- Level N

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- internal k-v layout

- MEMORY
- DISK
- pages
- SSTables
- leveled
- tiered
- sorted

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- internal k-v layout
size ratio
merge policy
filters bits per entry
size of buffer/cache
internal k-v layout
DOMAIN?

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- internal k-v layout
Domain? AMPLIFICATION?

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- internal k-v layout

Read
Update
Memory
LSM-trees

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- internal k-v layout
LSM-trees
B-trees
Logs
Arrays
Bitmaps

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- key retention
- value retention
- partitioning
- sub-block links
- fanout
key retention
value retention
partitioning
sub-block links
fanout
unified design space
POSSIBLE NODE DESIGNS
POSSIBLE STRUCTURES

POSSIBLE NODE DESIGNS
STARS IN THE SKY

10^{24}

POSSIBLE DATA STRUCTURES

10^{32}, 2-node
10^{48}, 3-node
TIGRIS specifications are orders of magnitude shorter than high-performance container implementations written in traditional systems languages like C, while being retargetable across platforms and allowing for sophisticated automatic optimization and tuning. We demonstrate the power of TIGRIS, a domain-specific language and compiler, by describing an optimizing compiler that compiles TIGRIS specifications to highly efficient container code. We present initial results that indicate that TIGRIS can express and optimize container data structures in a concise manner, without sacrificing performance. While high-performance container structures are notoriously difficult to write and typically hard to maintain and debug, our separation by introducing a declarative domain specific language allows us to quickly iterate between alternative designs, and to use concise descriptions and the functional feature descriptions. We also show how the concise descriptions in TIGRIS can be used to express and optimize container data structures.
The TIGRIS Container Description Language and Compiler specifications to highly-evaluated
This not only makes the code hard to maintain and debug, workloads.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00. permission and/or a fee.
republish, to post on servers or to redistribute to lists, requires prior specific
bear this notice and the full citation on the first page. To copy otherwise, to
not made or distributed for profit or commercial advantage and that copies
Permission to make digital or hard copies of all or part of this work for

1. INTRODUCTION

Container design space.
reuse of optimizations can facilitate rapid exploration of the
formance. We also show how concise descriptions and the
data structures in a concise manner, without sacrificing per-
a large variety of traditional and special-purpose container
for sophisticated automatic optimization and tuning. We
C, while being retargetable across platforms and allowing
implementations written in traditional systems languages li
by describing an optimizing compiler that compiles TIGRIS

Figure 1.

-5K since the dawn of CS

10^24

10^32, 2-node
10^48, 3-node

10^48-5x10^3 = 10^48 zero progress?